Refactoring to Microservice Architecture

Dr. Latha Sadanandam,

Software Architect,

Danske IT and Support Services India Pvt Ltd.,

Abstract

With the increasing usage of smartphones and other devices, digitization of banking sector is expected to catch up the increasing expectations of the customer. Banks have a significant role in our lives. Every one of us will execute at least a single financial transaction in a day. Hence, it becomes necessary for banks to enrich customer experience. Digitization becomes mandate feature for banks since it is being adopted in all industries in day to day life. Banks love mainframes because only mainframes can provide a single, unified, efficient solution to a host of different problems. Most of the banks uses Mainframe because of it robust, reliable and secured processing power. It also supports the new technologies like mobile, cloud etc., A business case is presented in this paper to explain Micro service and API framework for existing legacy system. Existing architecture is tightly coupled services with less standardization and fair performance. The aim of this paper is to provide solutions to convert the existing architecture to flexible service to support business for time-to-market, increase in performance and operational efficiency and improve customer experience.

1. Application Program Interface (API)

Application Program Interface (API) is used to communicate between programs, data exchange would be part of communication. A common structure has to be agreed between programs for smoother communication. Programs at both end has to know the location of each other and the structure of the data to be exchanged in advanced which is tightly coupled.

A revolution happened by introduction of Service Oriented Architecture framework. In SOA Era, Programs were loosely coupled and data transfer mechanism was done using XML using Web API. Extensible Markup Language (XML) is language and platform independent. This covered the way for producing and exposing APIs over network with better business enablement capabilities including request access, entitlement, identification, authorization, management, monitoring, and analytics. The standard format of data structure (XML Payload) was huge and heavy to be exchanged over network protocol.

A new mechanism was required which is lighter version of XML (SOAP message used in SOA architecture). So Representational State transfer (REST) was introduced. REST is an architectural style of any interface between systems using HTTP to obtain data and generate operations on those data in all possible formats, such as XML and JSON. It has actions to be taken on specific IT resources (File, Image, Database, Service etc.,) using HTTP/HTTPS protocol based on event triggered. REST verbs used with the protocol are POST, GET, PUT and DELETE.

Since HTTP Protocol is used, URL can be used to access the resources. JSON has quickly become the format of choice for REST APIs. It has a lightweight, readable syntax that can be easily manipulated. The standard request and response to and from API would be in form of JSON.

Benefits of REST API are not limited to

- Improves the portability of the interface to other types of platforms and allows the different components of the developments to be evolved independently.
- Increases the scalability of the server application without any disruption since the client and server are loosely coupled.
- Increase in productivity and faster time-to-market.

2. Microservice

Monolithic architecture is traditional server-side systems. The entire system's function is based on a single application. Application can be created with basic features and then scale up over time.

While monolithic architecture puts all of the functions into a single process, Microservice applications break those processes into individual functions. Multiple services are build inside of the application. Each can be tested and developed individually. The services will run as separate processes.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

Microservice is a form of Service Oriented Architecture (SOA) style of developing loosely coupled and fine-grained software application as independent deployable services. Each service runs as a unique process and communicates through lightweight protocol to serve a well-defined business function. It allows individual service for continuous enhancement without any disruption and supports continuous delivery and deployment.

3. Refactoring to Microservice Architecture

Refactoring to Microservice architecture also does not mean to throw away existing application and rewrite new micro services. Rather in majority cases, it is incredible to throw away the application. A methodology has to be identified on how to refactor an existing application using Microservice.

A business capability is a concept from business architecture modelling. It is something that a business does in order to generate value. A business capability often corresponds to a business object. Hence break down monolithic application to services corresponding to business capability. Identifying business capabilities and hence services requires an understanding of the business. This method is stable since the business capabilities are relatively stable.

Once the business capabilities are identified, all the capabilities/Functions cannot be refactored to service. Each service can be built in any technology, since the services are independent form the other and run as separate process. This technology independence enables gradual migration to new modern technologies. But still it is not mandate to replace/migrate the entire application. Legacy code which is robust, secure and highly available can still be maintained in the mainframe system. Mainframe system provides stable solution for security, handling high volume with robust throughput with support of structured data and handling optimized batch workloads to process the data.

Demarcate the business functions as “True core and core surround”. “True core” functions are core capability of the system, for example in core banking system creation of account, Ledger maintenance, interest calculation for accounts etc., can be retained in the legacy system. “Core Surround” functions can be sub activity of the core system such as payment validation, charge calculation. All these activities can be separated from the core capabilities and migrated to new platform / product.

“True core” functions are exposed as Mainframe services and gradually transform the core functions to the newer technologies. This pattern is termed by Martin Fowler as “Strangler Application Pattern”. This approach is used in transformation strategy of legacy modernization.

4. Microservice Design Patterns

Based on business requirement, functional decomposition is done which provides agility, flexibility and scalability. But the end point of business is to have an application which servers all the functionality. Orchestration of services are required to achieve a business functionality. Design patterns for orchestration are discussed in this section.

- Aggregator Microservice design pattern
- Proxy Microservice design pattern
- Chained Microservice design pattern
- Branch Microservice design pattern
- Shared Microservice design pattern
- Asynchronous messaging Microservice design pattern

4.1 Aggregator Microservice design Pattern

In this pattern, Aggregator act as orchestrator for the service. It might be a simple web page or composite service that invokes multiple services to achieve the functionality required by the application. Since all services are exposed using a lightweight REST services. The web page can retrieve the data and process/display it. If any processing (business logic) is required for the data retrieved from services, computing program (bean) can be developed that would apply the business logic and computation on the data to be displayed on the web page.

An advantage of this pattern is that the individual services can evolve independently and the business need is still provided by the composite microservice.

4.2 Proxy Microservice design pattern

In this case, no aggregation might be required, but a different microservice can be invoked based upon the business need. The proxy may be a *dumb proxy* in which case it just delegates the request to one of the services.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

Alternatively, it may be a *smart proxy* where some data transformation is applied before the response is served to the client. An example of this would be where the presentation layer to different devices can be encapsulated at the proxy level.

4.3 Chained Microservice design pattern

Chained microservice design pattern produce a single consolidated response to the request. The key feature is that the client is blocked until the complete chain of request/response. The synchronous nature of the chain will make client to wait for long time.

Advantage of this pattern is any service can be removed or added as the per the business requirement.

4.4 Branch Microservice design pattern

Branch microservice design pattern extends Aggregator design pattern and allows simultaneous response processing from multiple chains of microservices. Service either a web page or a composite microservice, can invoke two different chains concurrently in which case this will resemble the Aggregator design pattern. Alternatively, Service can invoke only one chain based upon the request received from the client.

4.5 Shared Microservice design pattern

In this design pattern, some microservices might share caching and database stores. This would only make sense if there is a strong coupling between the two services. This might be a business needs to use either the database or caching.

This act as anti-pattern of microservice because of tight coupling and sharing of resources across services.

4.6 Asynchronous messaging Microservice design pattern

A blend of REST call and pub/sub messaging may be used to realize the business need in this pattern. In this design pattern, Service may call a service synchronously which might then communicate with Service asynchronously using a shared message queue.

This pattern has the limitation of being synchronous and the response time is slow.

5. Architecture framework for exposing Legacy application as REST services

Most often the demarcated “True core” functions resides on Mainframe. The best way is to expose these applications as REST services/API in digital era for the customer touch points like Mobile etc., to maximize the value of the system.

There are multiple models to perform REST on mainframe. Let us discuss here on three architectural model.

- Exposing REST calls at Mid-layer (Data power Appliance)
- Exposing REST calls in CICS by exposing applications as JSON web service
- Handling REST calls using Gateway

Let us discuss on the architecture of these three models with its pros and cons.

5.1 Exposing REST calls at Mid-layer

In this model, the REST API call can be handled in a mid-layer device such as Datapower or IIB. The rest resource, CICS web service (non-REST) is processed in Mainframe system and the response is provided for the request. The protocol could be Messaging protocols (MQ, JMS etc.,). It has less changes on mainframe legacy system. But the drawback is that mapping between legacy system data structure and requester data structure has to be handled in mid-layer manually by using wrappers.

This architecture model uses Asynchronous messaging Microservice design pattern which has its own limitation of response latency and not being synchronous.

5.2 Exposing REST calls in CICS by exposing applications as JSON web service

REST call can be made directly to CICS. CICS application has to be exposed as JSON Web service, configuring and deploying the services in CICS transaction region. A proxy is required to access the services in the CICS region of legacy system.

CICS JSON Web service can be achieved using two different approaches – Top-down and Bottom-up.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

In Top-down approach, using JSON schema from client is used to create copybook using DFHJS2LS utility. The copybook can be used in the wrapper program to map the structure to CICS source program.

In Bottom-Up approach, JSON web service are created using request and response copybooks using DFHLS2JS utility.

This pattern is a direct way of invoking CICS services, but requires different methods for each type of CICS application.

Proxy microservice design pattern is applied to this architecture, In this case, no aggregation might be required, but a different microservice can be invoked based upon the business need. Proxy help in data transformation before the response is served to the client. The response can accommodate request from any type of client. The encapsulation is done at the proxy layer.

5.3 Handling REST calls using Gateway

Using Gateway in LPARs to handle the REST API calls. The gateways provide controllable entry point for LPARs (e.g., Z/OS connect). Z/OS connect Enterprise Edition runs on liberty profile on zos. It act as receiving server for URL, parse the URL and identify the resource. Z/OS connect EE is configured such that identified resources is mapped to a CICS program on a specific region.

But CICS handles the I/O structure in COMMAREA/CHANNELS copybook structure, Z/OS connect EE has provision to convert JSON request to COMMAREA/CHANNELS copybook structure and vice versa.

Execute the CICS program based on the query parameter and return back the output copy book structure back to Z/OS connect EE which converts in JSON Payload.

This pattern is standard approach, since auditing and tracing is easy and controls the call based on security criteria.

Aggregator or chain or branch microservice design pattern is applied in this framework. This patterns helps the model to adopt a solution, that the individual services can evolve independently and the business need is provided by the composite microservice.

6. Conclusion

A microservices architecture breaks down application components into small, manageable services, each running within its own function independently by a different team. This makes it easier to change functions and qualities of the system at any time. Microservices lends itself to continuous delivery software development, because a change to a small part of the application only requires one or a small number of services to be redeployed. The services can be designed and developed as completely autonomous entities, with individual services for new applications being reused or integrated with third-party services without affecting other applications.

With microservices-based architecture, implementation can be done using any technologies and frameworks. Each microservice can be built by any set of languages or tools, since each is independent from the others and each runs a separate process. This technology independence also means that individual services within an application can be gradually replaced with applications based on more modern technologies—without having to replace the entire application. Microservices are independent and agile with rapid application evolution. Organizations can respond more quickly to customer and market feedback, and releases no longer need to be delayed by the schedule of a single release.

Acknowledgement

The author would like to thank the **Management team, Danske IT and support services India Pvt ltd.,**for providing support and providing permission to publish the paper.

The author would also like to acknowledge and thank the authors and publishers of referenced papers for making available their invaluable work which served as excellent input for this paper.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500